

A Bayesian Approach to Predicting NFL Quarterback Scores in Fanduel Tournaments

STAT 578, Fall 2017, Team 5: Aaron Ray, Kiomars Nassiri, Michael Chan

October 25, 2017

Project Description

The National Football League (NFL), being one of the major professional sports leagues in North America, has a wide audience. participates in the NFL craze by competing in fantasy football tournaments organized by the daily fantasy site, “FanDuel.com”. Participants in a **Fantasy Football** game act as the managers of a virtual football team and try to maximize their points by picking up the best line-up. Points are given based on actual performance of players in real-world competition. For the purpose of this project we have chosen to work with the data gathered from the **FanDuel** internet company. We will leverage a Hierarchical Bayesian approach with the Markov Chain Monte Carlo method to predict the fantasy points likely to be scored by an NFL quarterback in any given game. The goal is to predict the points scored by each player given certain prior conditions and predictor variables that will assist our model in providing credible posterior prediction intervals.

The analysis is inspired by the study presented in the article, **Bayesian Hierarchical Modeling Applied to Fantasy Football Projections for Increased Insight and Confidence**, by Scott Rome.

Team Members

- **Aaron Ray** (aaronwr2@illinois.edu)*
- **Kiomars Nassiri** (nassiri2@illinois.edu)
- **Michael Chan** (mhchan3@illinois.edu)

*Contact Person

Dataset Description

Team has set up a process to gather the historical data from the RotoGuru website. The following is the code used to get the data from RotoGuru:

```
# Scrape rotoguru1 site for weekly FanDuel stats and bind each week's data to the
# pre-defined dataframe, 'd'.

for(year in 2014:2017){
  for(week in 1:16){
    page = read_html(
      gsub(" ", "", ,
           paste("http://rotoguru1.com/cgi-bin/fyday.pl?week=", week, "&year=",
                 year, "&game=fd&scsv=1"))
    )
    dtext = page %>% html_nodes("pre") %>% html_text(trim = TRUE)
    dtable = read.table(text=dtext, sep = ";", header=TRUE, col.names = cnames,
```

```

    quote=NULL)
d = rbind(d,dtable)
}
}

```

Data cleaning is performed using R routines. Some data cleaning tasks are needed to calculate Player rank.

Response Variables

- **FanDuelPts**: Points position at the end of a single game

Predictor Variables

- **AvgPts5Wks**: The 5 game average points of the player
- **AvgOppPAP7Wks** : The 7 game average Opposing Points Allowed to Position (OppPAP) by the current player's opposing defense. For example, if the Buffalo Bills defense allowed a total of 30 points per game to wide receivers for six games straight, then this number would equal to the average of 30 for any wide receiver facing the Bills defense.
- **Position**: The position the player plays
- **HomeGame**: Whether it is home game.
- **Rank**: The rank of a player based on recent performance

Analysis Ideas

Model

At the lowest level, we model the performance (**FanDuelPts**) as normally-distributed around a true value:

$$y|\alpha, \beta_{defense}, \beta_{home}, \beta_{away}, \sigma_r^2 \sim N(\alpha + X_{defense} \cdot \beta_{defense} + X_{home} \cdot \beta_{home} + X_{away} \cdot \beta_{away}, \sigma_y^2 I)$$

where

α = The average fan duel point of the previous 5 weeks of the player, **AvgPts5Wks**

$\beta_{defense,p}$ = defense coefficient against team t for position p

$\beta_{home,p,r}$ = home coefficient for position p and a rank r player

$\beta_{away,p,r}$ = Away coefficient for position p and a rank r player

y = **FanDuelPts**

x_p = interaction indicator term for opposing team score allowed by position p

$x_{home,p,r}$ = interaction indicator term for rank r, position p, and whether it is home game

At higher level, we model the defense effect, $\beta_{defense}$, as how good(bad) a particular team's defense is against the player's position. We pool the effect based on the position of the player. That is, the defense coefficient is normally distributed from the same position specific distribution.

$$\beta_{defense,p} \sim N(\delta_p, \sigma_\delta^2)$$

where σ_δ is constant = 1000

For the home and away game effect, β_{home} and β_{away} , we model the effect for player of the same rank has the same distribution. We model the home and away game effect to be the same for players of the same position.

$$\beta_{home,p,r} \sim N(\eta_r, \sigma_\eta^2)$$

$$\beta_{away,p,r} \sim N(\rho_r, \sigma_\rho^2)$$

where σ_η, σ_ρ are constant = 1000

We will approximate non informative prior using:

$$\sigma_y \sim Inv - gamma(0.0001, 0.0001)$$

$$\delta \sim N(0, 10000^2)$$

$$\eta \sim N(0, 10000^2)$$

$$\rho \sim N(0, 10000^2)$$

Here is the JAGS model:

```
#sink("fdp.bug")
#cat("
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(alpha[i] + inprod(X.defense[i, ], beta.defense)
                  + inprod(X.home[i, ], beta.home)
                  + inprod(X.away[i, ], beta.away), sigmasqinv)
  }

  # The entry of the beta.defense corresponds to Opponent:Position
  # In our model, we pool the beta.defense based on position.
  # i.e. All defense effects of the same position are drawn from the same distribution
  for (p in 1:Num.Position) {
    for (f in 1:Num.fixed.pred) {
      beta.defense[(f-1) * Num.Position + p] ~ dnorm(delta[p], 1/1000^2)
      delta[(f-1) * Num.Position + p] ~ dnorm(0, 1/100000^2)
    }
  }

  # The entry of the beta.home and beta.away corresponds to Rank:Position
  # In our model, we pool the beta.home/away based on rank
  for (r in 1:Num.Rank) {
    for (t in 1:Num.Position) {
      beta.home[(t-1) * Num.Rank + r] ~ dnorm(eta[r], 1/1000^2)
      beta.away[(t-1) * Num.Rank + r] ~ dnorm(rho[r], 1/1000^2)
    }
    eta[r] ~ dnorm(0, 1/100000^2)
    rho[r] ~ dnorm(0, 1/100000^2)
  }

  sigmasqinv ~ dgamma(0.0001, 0.0001)
  sigmasq <- 1/sigmasqinv
}
#   ",fill = TRUE)
#sink()

library(knitr)
```

Sample Data

```
fdp <- read.csv("fdpfinal.csv", sep = ',', header = TRUE)

head(fdp)
```

```
##   Position Year YearWeek Opponent Week PlayerId          Name      Team
## 1      QB 2015     201513  Steelers  13    1060 Hasselbeck, Matt  Colts
## 2      QB 2015     201514  Jaguars  14    1060 Hasselbeck, Matt  Colts
## 3      QB 2015     201515  Texans   15    1060 Hasselbeck, Matt  Colts
## 4      QB 2015     201516 Dolphins 16    1060 Hasselbeck, Matt  Colts
## 5      QB 2015     201501  Ravens   1    1081 Manning, Peyton Broncos
## 6      QB 2015     201502 Chiefs    2    1081 Manning, Peyton Broncos
##   HomeGame FanDuelPts FanDuelSalary AvgOppPAP7Wks SdOppPAP7Wks OallAvgPAP
## 1          0       6.86        6500      20.95      8.045    17.44
## 2          0       9.08        6600      24.16      6.738    17.44
## 3          1       8.98        6400      16.36      9.690    17.44
## 4          0       3.96        6000      20.46      6.002    17.44
## 5          1       5.90        9100      18.99     11.697    17.44
## 6          0      21.24        8200      13.85      4.342    17.44
##   OallStddevPAP AvgPts5Wks StdevPts5Wks OffRnk5Wks DefRnk7Wks
## 1      2.909     14.85      4.824    Rank4      Rank1
## 2      2.909     14.82      4.897    Rank4      Rank1
## 3      2.909     13.56      5.490    Rank4      Rank3
## 4      2.909     12.11      5.566    Rank4      Rank2
## 5      2.909     14.96      8.496    Rank3      Rank1
## 6      2.909     10.53      5.011    Rank4      Rank4
```

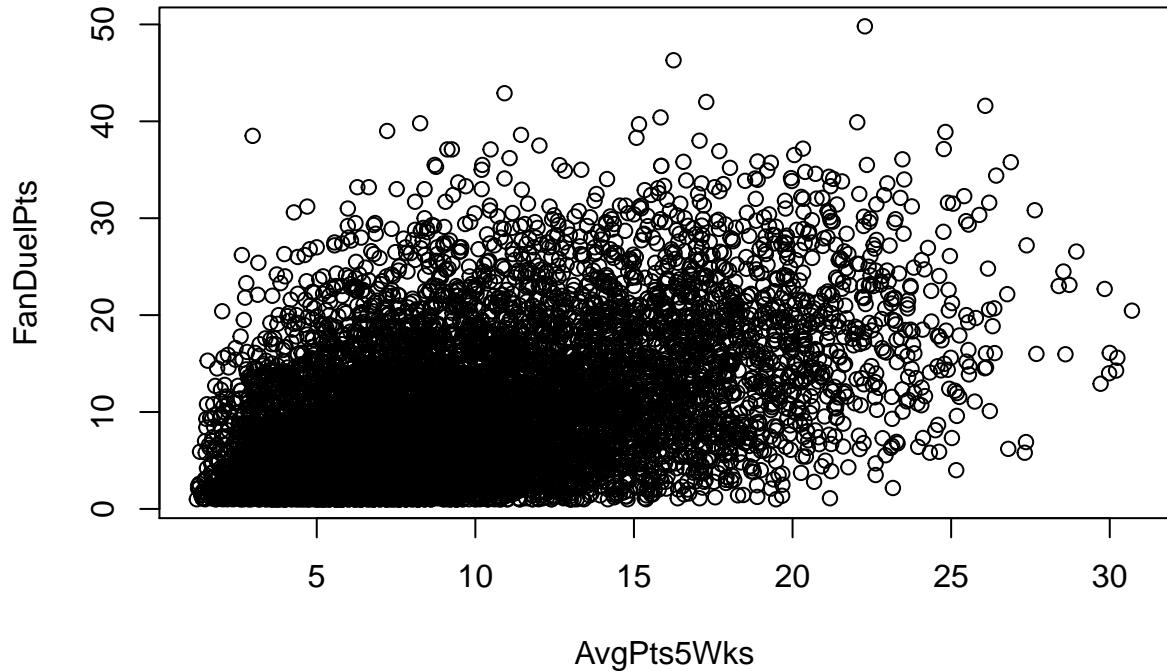
```
fdp['Rank'] = fdp$OffRnk5Wks
```

Simple Ideas

$$y|\alpha \sim N(\alpha, \sigma_y^2 I)$$

```
mod.classic = lm(FanDuelPts ~ AvgPts5Wks, data = fdp)

plot(FanDuelPts ~ AvgPts5Wks, data = fdp)
```



All over the place, let's add the team defense

X.defense is the indicator matrix

** Set up train data **

```
#fdp_train=fdp[fdp$Year == 2015, ]
fdp_train=fdp[fdp$Year == 2016 & ((fdp$Position == "QB"& fdp$FanDuelSalary > 6500 & !is.na(fdp$FanDuelSalary)) | (fdp$Position == "RB"& fdp$FanDuelSalary > 6500 & !is.na(fdp$FanDuelSalary)))
#fdp_train=fdp[fdp$Year == 2015 & (fdp$Position == "QB" | fdp$Position == "RB") & fdp$FanDuelSalary > 6500 & !is.na(fdp$FanDuelSalary)]
```

```
fdp_test = fdp_train[fdp_train$YearWeek >= 201615, ]
fdp_test = droplevels(fdp_test)
fdp_train=fdp_train[fdp_train$YearWeek < 201615, ]
fdp_train = droplevels(fdp_train)
```

```
Use.Rank = TRUE
Num.Opponent = length(unique(fdp_train[, "Opponent"]))
Num.Position = length(unique(fdp_train[, "Position"]))
#Num.fixed.pred=2 #AvgOppPAP7Wks + FanDuelSalary
Num.fixed.pred=1 #AvgOppPAP7Wks
if (Use.Rank) {
  Num.Rank = length(unique(fdp_train[, "Rank"]))
  Num.HomeAwayInit = Num.Rank
  Model.File.Ext = ""
} else {
  Num.HomeAwayInit = 1
  Model.File.Ext = ".norank"
```

```

}

if (Num.Position == 1) {
  #X.defense = model.matrix(~ 0 + AvgOppPAP7Wks + FanDuelSalary, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks, data=fdp_train)
  if (Use.Rank) {
    X.home = model.matrix(~ 0 + Rank , data=fdp_train)
    X.away = model.matrix(~ 0 + Rank , data=fdp_train)
  } else {
    X.home = rep(1, nrow(fdp_train))
    X.away = rep(1, nrow(fdp_train))
  }
} else {
  #X.defense = model.matrix(~ 0 + AvgOppPAP7Wks:Position + FanDuelSalary:Position, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks:Position, data=fdp_train)
  if (Use.Rank) {
    X.home = model.matrix(~ 0 + Rank:Position , data=fdp_train)
    X.away = model.matrix(~ 0 + Rank:Position , data=fdp_train)
  } else {
    X.home = model.matrix(~ 0 + Position , data=fdp_train)
    X.away = model.matrix(~ 0 + Position , data=fdp_train)
  }
}

X.home = X.home * fdp_train$HomeGame
X.away = X.away * (1- fdp_train$HomeGame)
X = cbind(X.defense, X.home, X.away)

```

```

library(rjags)
set.seed(20171008)

```

```

# Initialization List for the 4 chains
jags.inits=list(
  list( sigmasqinv= 0.01, delta = rep(-100000, Num.Position * Num.fixed.pred),
        eta = c(100000, -100000, 100000, -100000)[1:Num.HomeAwayInit],
        rho = c(-100000, 100000, -100000, 100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 ),
  list( sigmasqinv= 0.01, delta = rep(100000, Num.Position * Num.fixed.pred),
        eta = c(100000, -100000, -100000, 100000)[1:Num.HomeAwayInit],
        rho = c(-100000, 100000, 100000, -100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 1 ),
  list( sigmasqinv=0.000001, delta = rep(-100000, Num.Position * Num.fixed.pred),
        eta = c(-100000, 100000, -100000, 100000)[1:Num.HomeAwayInit],
        rho = c(100000, -100000, 100000, -100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 2 ),
  list( sigmasqinv=0.000001, delta = rep(100000, Num.Position * Num.fixed.pred),
        eta = c(-100000, 100000, 100000, -100000)[1:Num.HomeAwayInit],
        rho = c(100000, -100000, -100000, 100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 3 )
)

```

```

data.jags <- list(
  y= fdp_train$FanDuelPts,
  alpha = fdp_train$AvgPts5Wks,
  X.defense = X.defense,
  X.home = X.home,
  X.away = X.away,
  Num.fixed.pred=Num.fixed.pred,
  Num.Position=Num.Position,
  #Num.Opponent=Num.Opponent,
  Num.Rank=Num.Rank
)

burnAndSample = function(m, N.burnin, N.iter, show.plot, mon.col, n.thin=1) {
  update(m, N.burnin) # burn-in

  x <- coda.samples(m, mon.col, n.iter=N.iter, n.thin)

  if(show.plot) {
    plot(x, smooth=FALSE)
  }

  gelman.R = gelman.diag(x, autoburnin=FALSE, multivariate = FALSE)
  print(gelman.R)

  result <- list(
    coda.sam = x,
    gelman.R.max=max(gelman.R$psrf[, 1])
  )

  return(result)
}

runModel=TRUE
runSample=TRUE

mon.col <- c("delta", "eta", "rho", "beta.defense", "beta.home", "beta.away", "sigmasq")

NSim = 30000
NChain = 4
NThin = 5
NTotalSim = NSim * NChain / 5
if (runModel) {
  m <- jags.model("fdp.bug", data.jags, inits = jags.inits, n.chains=NChain, n.adapt = 1000)
  save(file=paste("fdp.jags.model.init", Model.File.Ext, ".Rdata", sep=""), list="m")
} else {
  load(paste("fdp.jags.model.init", Model.File.Ext, ".Rdata", sep=""))
  m$recompile()
}

## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:

```

```

##      Observed stochastic nodes: 746
##      Unobserved stochastic nodes: 29
##      Total graph size: 18345
##
## Initializing model

load.module("dic")

## module dic loaded

N.Retry.Loop = 1
if (runSample) {
  N.burnin=2500/2
  for (loopIdx in 1:N.Retry.Loop) {
    (start_time <- Sys.time())
    (N.burnin = N.burnin * 2)
    result = burnAndSample(m, N.burnin, NSim, show.plot=FALSE, mon.col = mon.col, n.thin=NChain)
    (end_time <- Sys.time())
    (result$gelman.R.max)
  }
  run.params = paste(".", N.burnin, ".", NChain, ".", NSim, ".", NThin, sep="")
  save(file=paste("fdp.jags.samples", run.params, Model.File.Ext, ".Rdata", sep=""), list="result")
  save(file=paste("fdp.jags.model", run.params, Model.File.Ext, ".Rdata", sep=""), list="m")
} else {
  N.burnin=2500/2 * (2**N.Retry.Loop)
  run.params = paste(".", N.burnin, ".", NChain, ".", NSim, ".", NThin, sep="")
  load(paste("fdp.jags.samples", run.params, Model.File.Ext, ".Rdata", sep=""))
  load(paste("fdp.jags.model", run.params, Model.File.Ext, ".Rdata", sep=""))

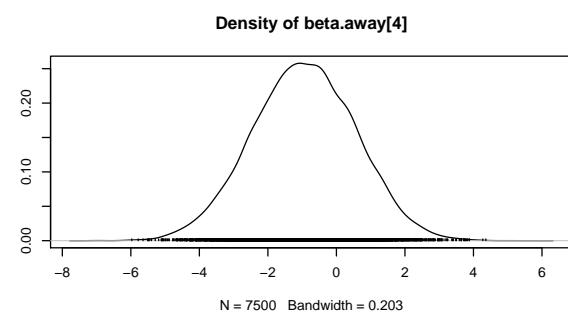
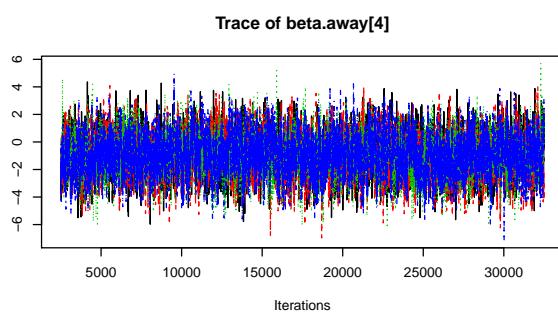
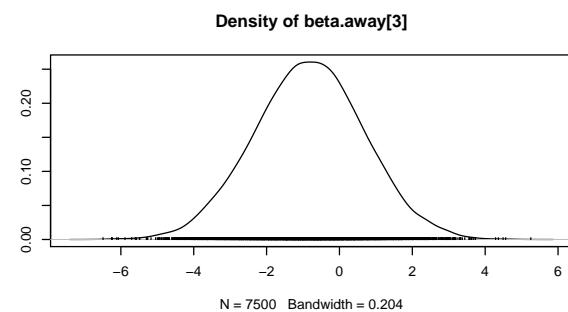
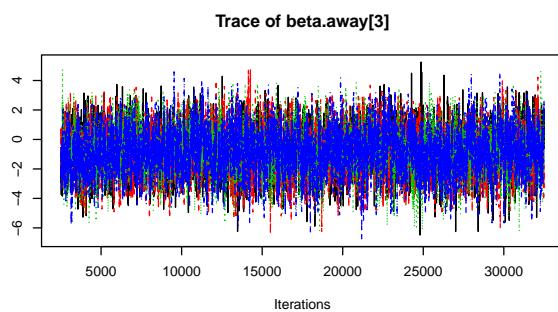
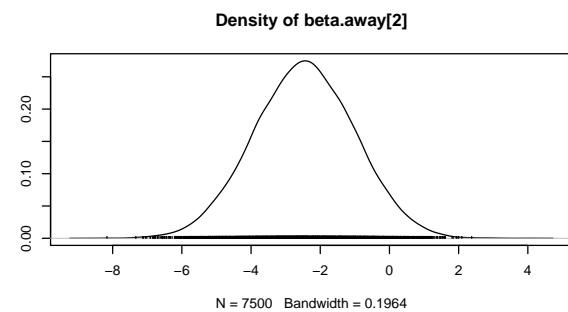
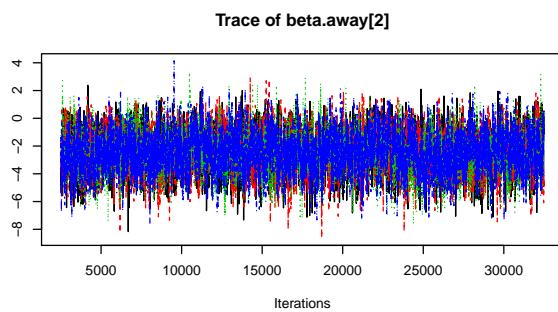
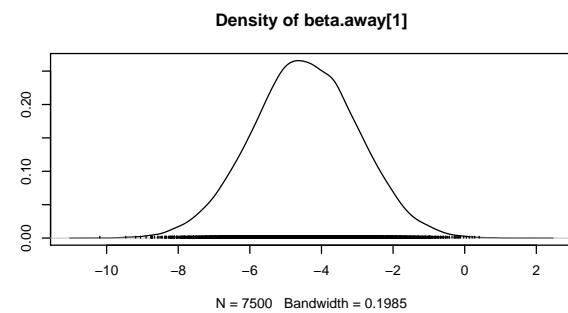
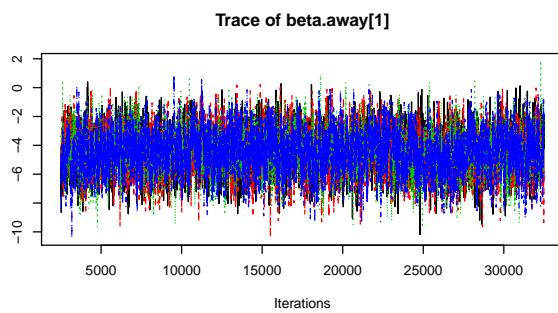
  gelman.diag(result$coda.sam, autoburnin=FALSE, multivariate = FALSE)
}

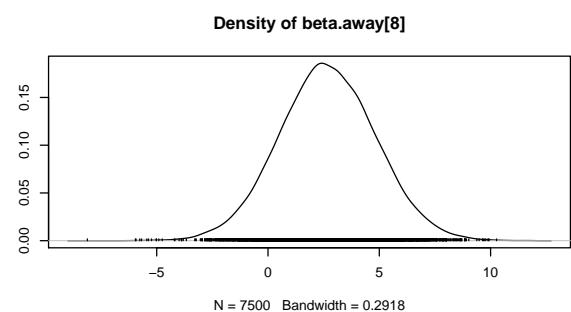
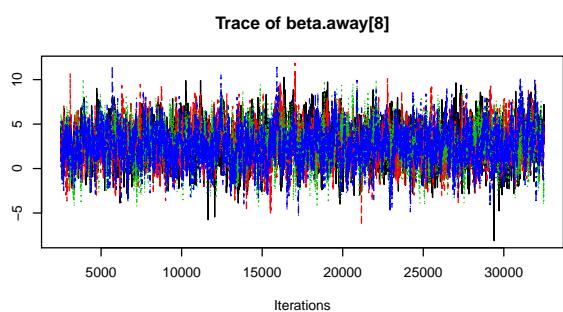
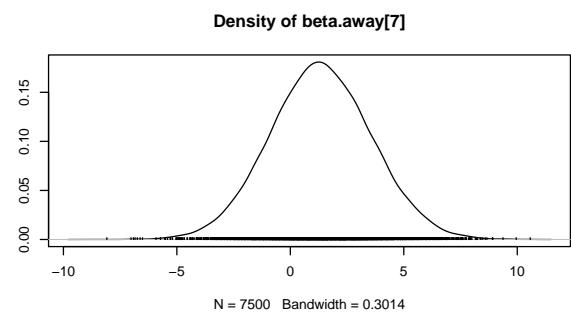
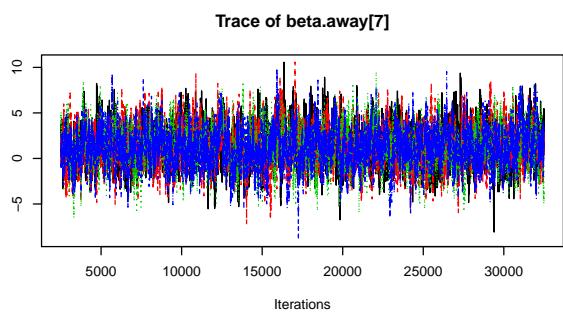
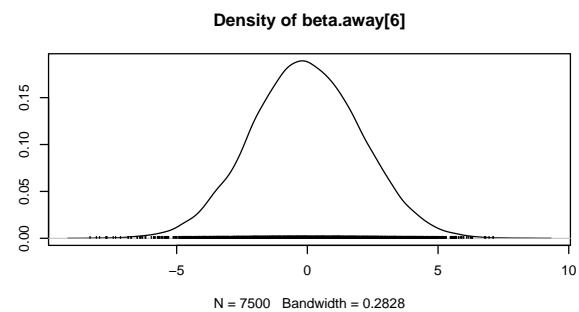
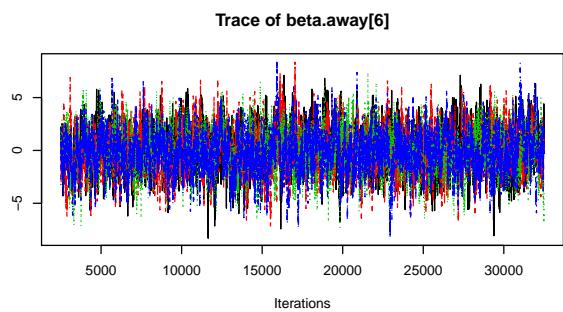
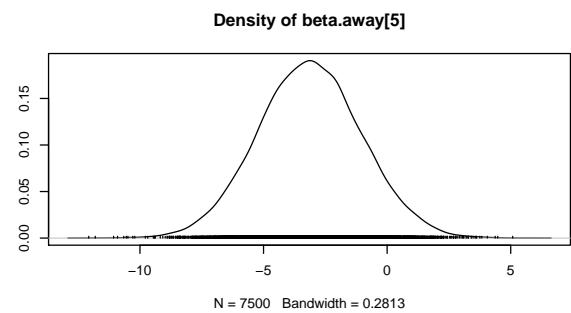
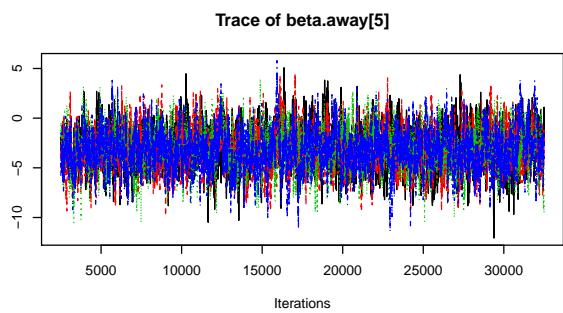
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## beta.away[1]      1      1.00
## beta.away[2]      1      1.00
## beta.away[3]      1      1.00
## beta.away[4]      1      1.00
## beta.away[5]      1      1.01
## beta.away[6]      1      1.01
## beta.away[7]      1      1.00
## beta.away[8]      1      1.01
## beta.defense[1]   1      1.00
## beta.defense[2]   1      1.01
## beta.home[1]      1      1.00
## beta.home[2]      1      1.00
## beta.home[3]      1      1.00
## beta.home[4]      1      1.00
## beta.home[5]      1      1.00
## beta.home[6]      1      1.01
## beta.home[7]      1      1.01
## beta.home[8]      1      1.01

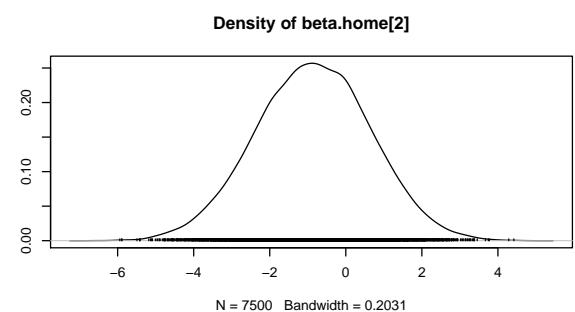
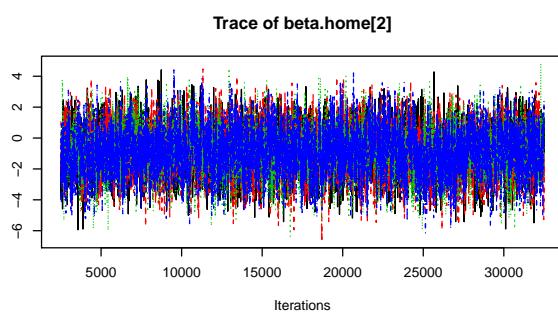
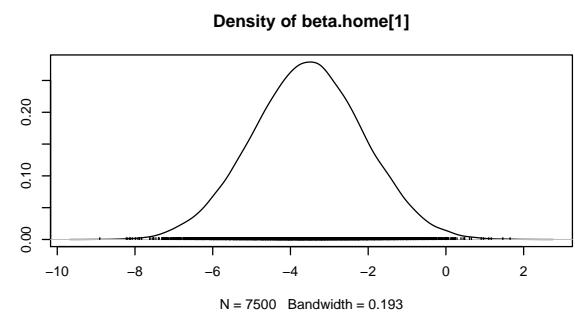
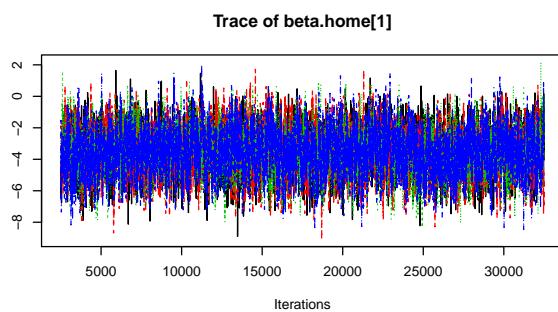
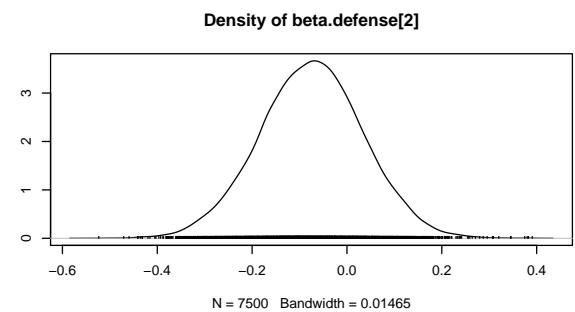
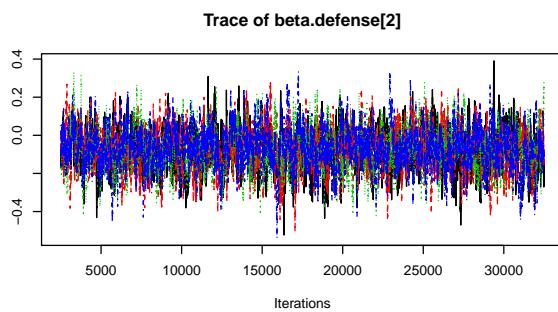
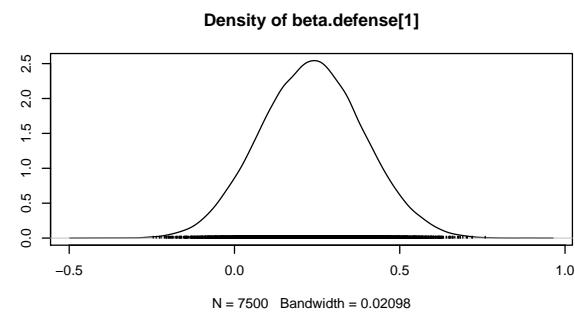
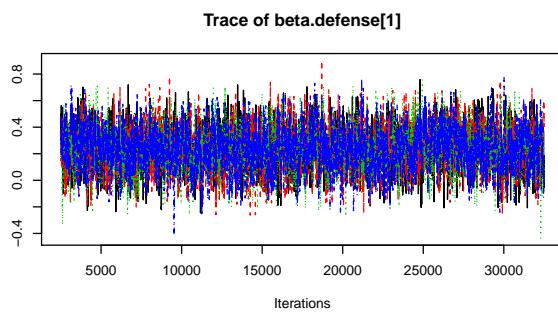
```

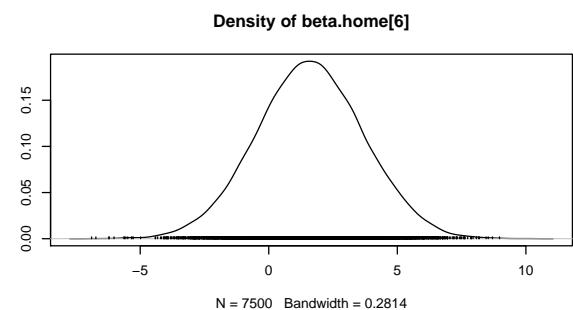
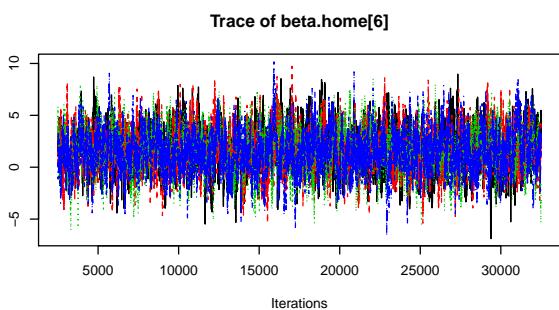
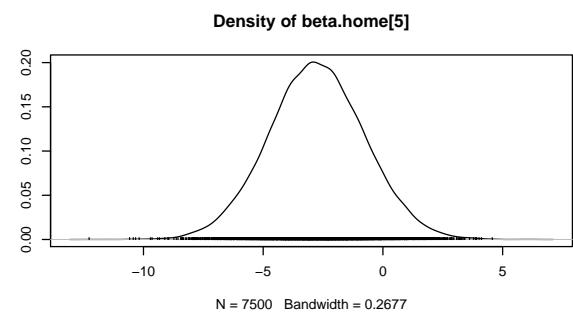
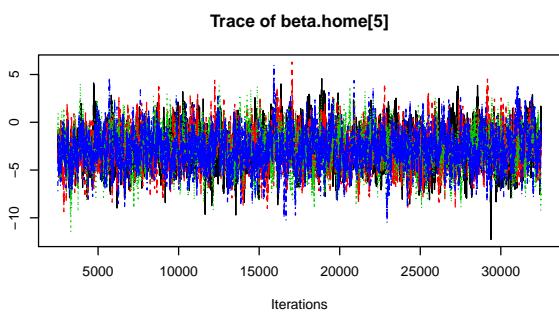
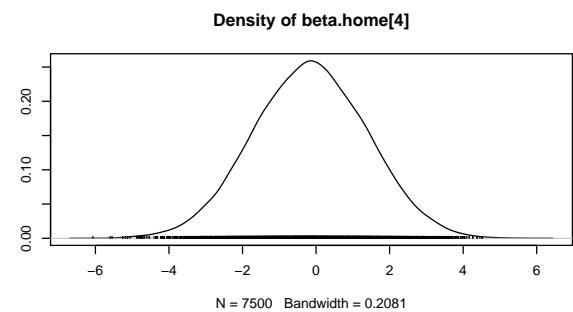
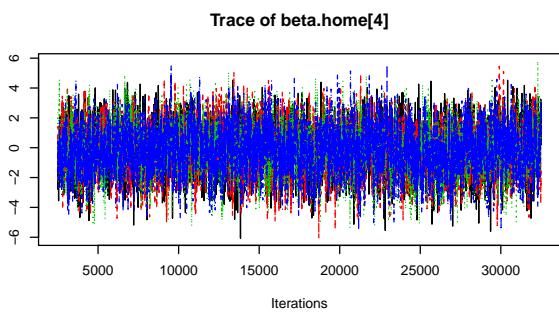
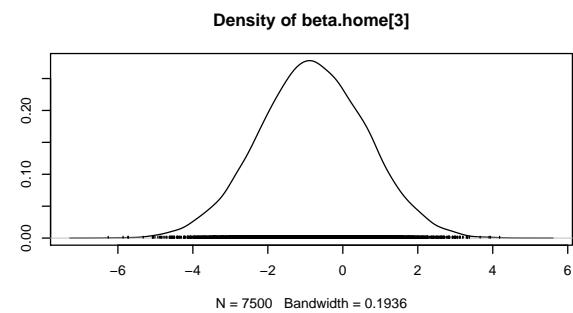
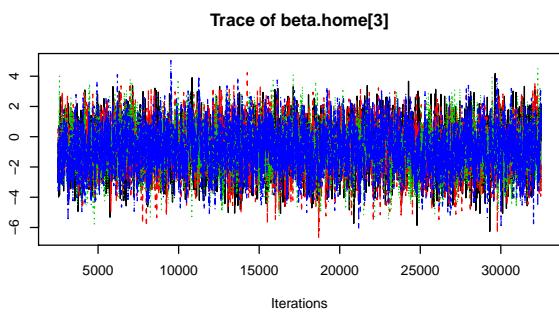
```
## delta[1]           1    1.00
## delta[2]           1    1.00
## eta[1]             1    1.00
## eta[2]             1    1.00
## eta[3]             1    1.00
## eta[4]             1    1.00
## rho[1]              1    1.00
## rho[2]              1    1.00
## rho[3]              1    1.00
## rho[4]              1    1.00
## sigmasq            1    1.00
```

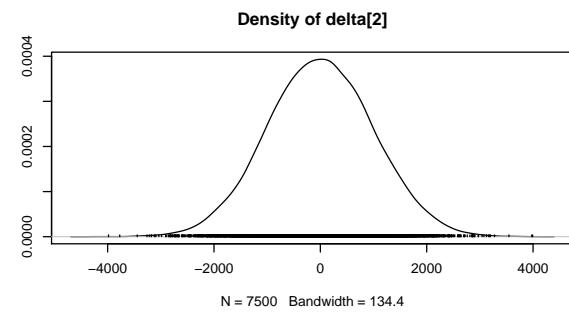
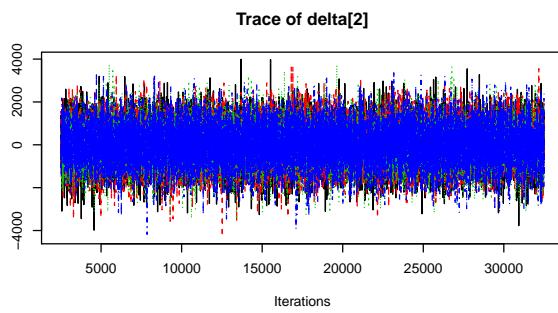
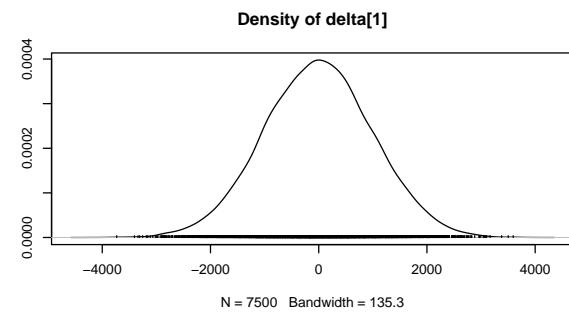
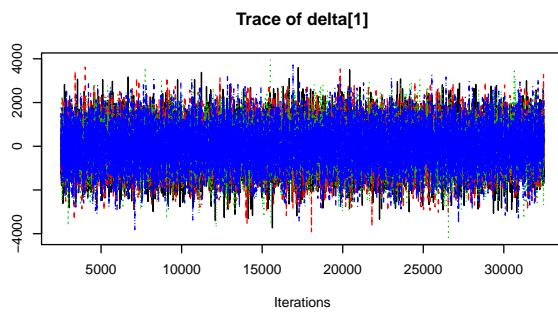
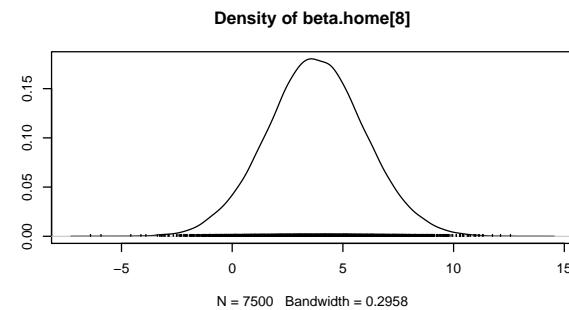
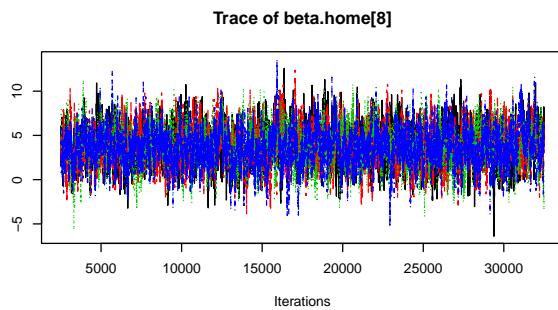
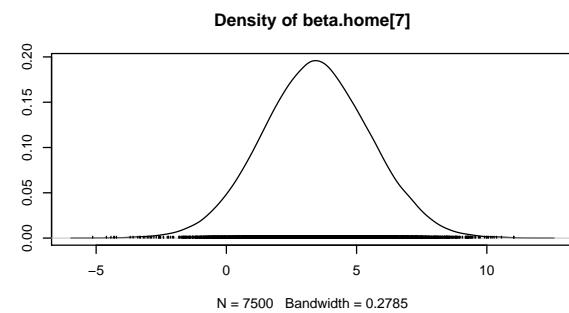
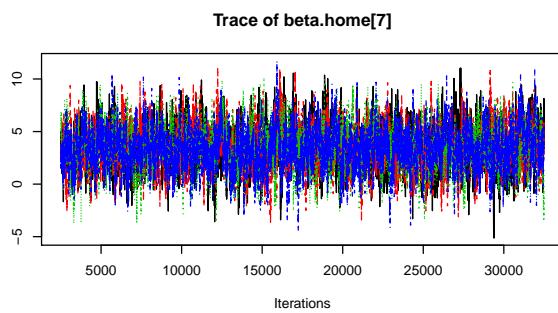
```
plot(result$coda.sam, smooth=FALSE)
```

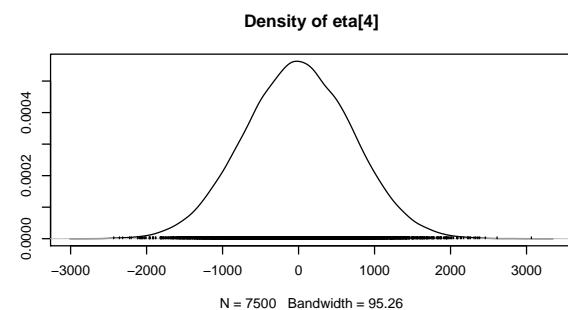
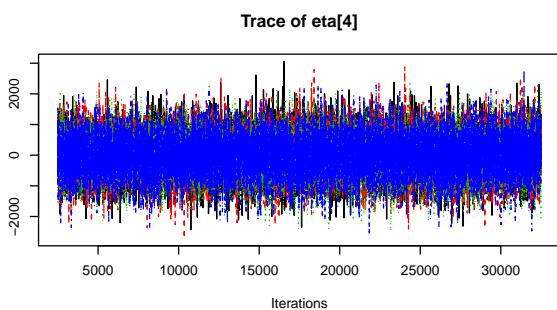
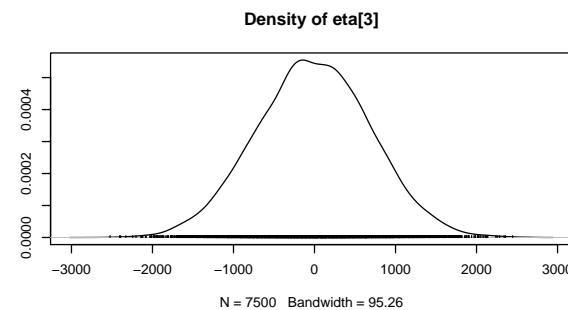
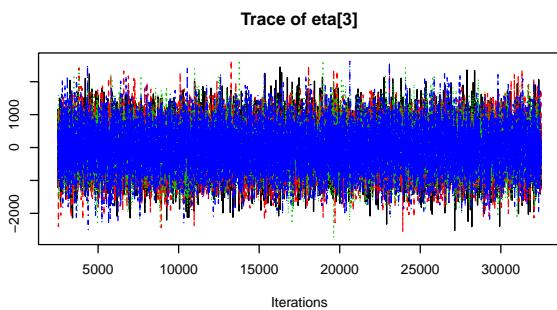
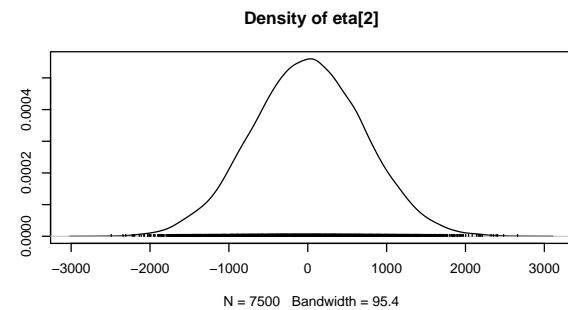
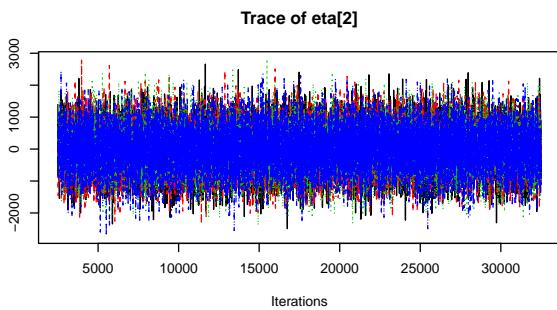
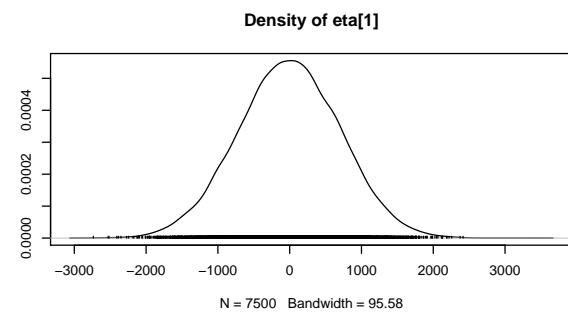
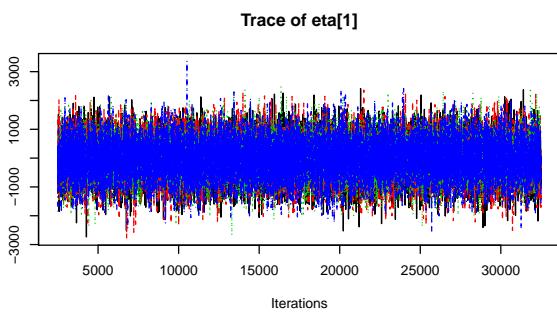


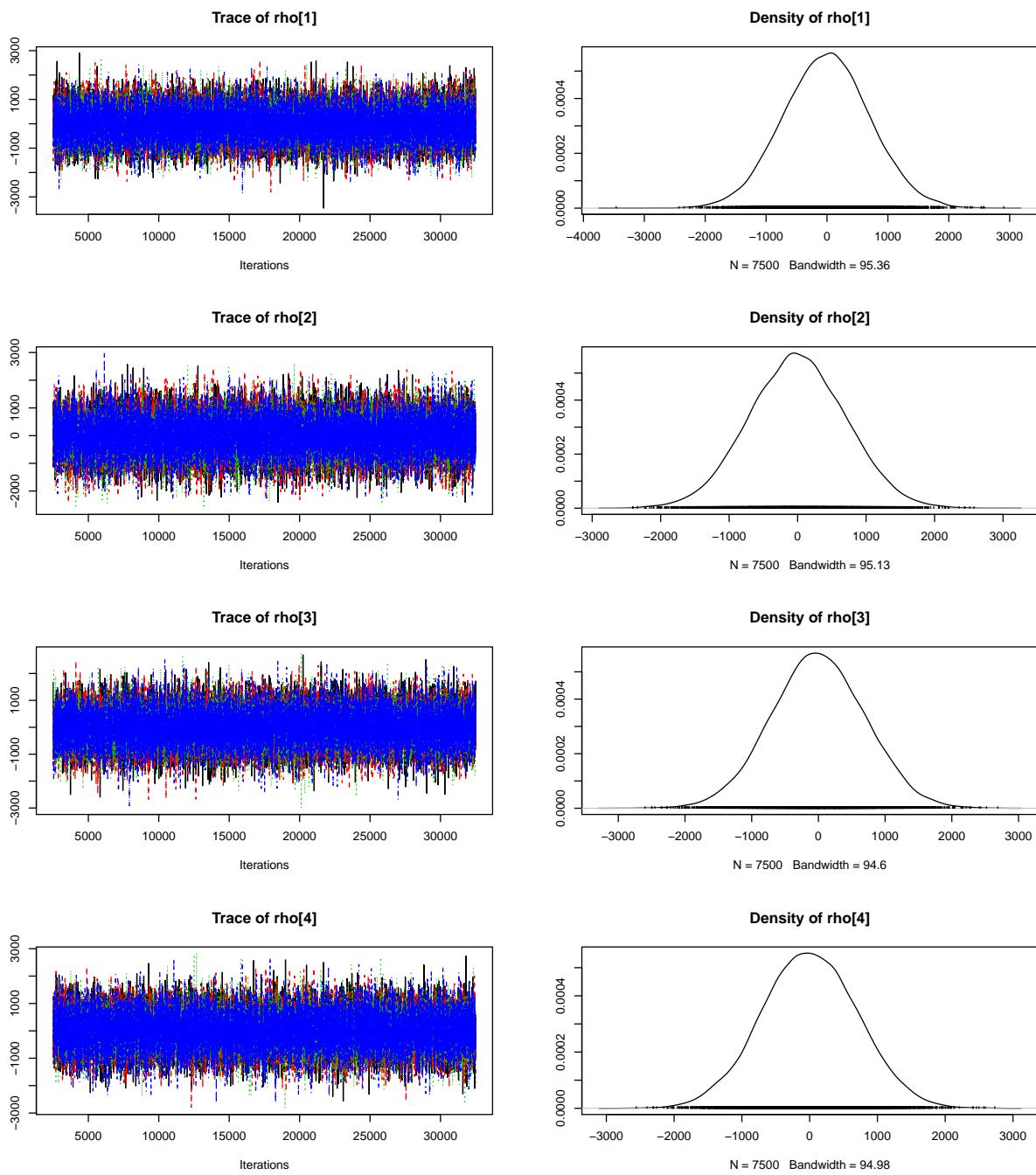


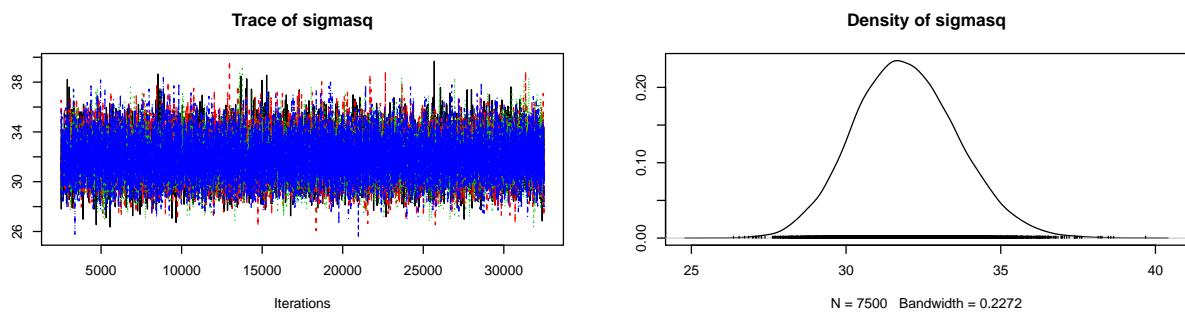












Converged as `gelman.R.max = 1.002 < 1.1` and the plot also looks good.

```
(m.summary = summary(result$coda.sam))
```

```
##
## Iterations = 2504:32500
## Thinning interval = 4
## Number of chains = 4
## Sample size per chain = 7500
##
```

```

## 1. Empirical mean and standard deviation for each variable,
## plus standard error of the mean:
##
##          Mean      SD Naive SE Time-series SE
## beta.away[1] -4.467  1.472 0.008499  0.02417
## beta.away[2] -2.461  1.456 0.008408  0.02444
## beta.away[3] -0.832  1.513 0.008733  0.02360
## beta.away[4] -0.942  1.505 0.008691  0.02398
## beta.away[5] -3.132  2.096 0.012100  0.05062
## beta.away[6] -0.051  2.097 0.012108  0.05033
## beta.away[7]  1.343  2.235 0.012903  0.05291
## beta.away[8]  2.670  2.175 0.012558  0.05066
## beta.defense[1] 0.235  0.156 0.000898  0.00285
## beta.defense[2] -0.075  0.109 0.000632  0.00283
## beta.home[1]   -3.547  1.431 0.008264  0.02330
## beta.home[2]   -0.859  1.506 0.008694  0.02329
## beta.home[3]   -0.805  1.435 0.008287  0.02291
## beta.home[4]   -0.162  1.543 0.008909  0.02516
## beta.home[5]   -2.767  2.006 0.011579  0.04770
## beta.home[6]   1.605  2.086 0.012045  0.04957
## beta.home[7]   3.449  2.074 0.011976  0.04888
## beta.home[8]   3.784  2.221 0.012826  0.05233
## delta[1]       -6.780 1003.390 5.793077  5.89868
## delta[2]        1.764 996.816 5.755122  5.75533
## eta[1]        -7.966 708.771 4.092093  4.08607
## eta[2]         2.218 707.396 4.084150  4.08699
## eta[3]        -9.512 706.400 4.078403  4.05872
## eta[4]        -1.529 706.367 4.078209  4.02323
## rho[1]         -6.188 707.088 4.082376  4.10605
## rho[2]        -3.301 705.424 4.072770  4.11118
## rho[3]        -1.886 701.548 4.050388  4.05048
## rho[4]         1.520 704.309 4.066333  4.04433
## sigmasq       31.947  1.685 0.009727  0.00974
##
## 2. Quantiles for each variable:
##
##          2.5%    25%    50%    75%   97.5%
## beta.away[1] -7.3717 -5.453 -4.4688 -3.46981 -1.611
## beta.away[2] -5.3105 -3.446 -2.4545 -1.46614 0.366
## beta.away[3] -3.8050 -1.845 -0.8234  0.18547 2.160
## beta.away[4] -3.8988 -1.965 -0.9387  0.09424 1.969
## beta.away[5] -7.2309 -4.542 -3.1370 -1.74709 1.026
## beta.away[6] -4.1378 -1.461 -0.0690  1.36573 4.063
## beta.away[7] -3.0093 -0.157  1.3282  2.84138 5.776
## beta.away[8] -1.5793  1.223  2.6498  4.12313 6.979
## beta.defense[1] -0.0663  0.129  0.2350  0.33995 0.543
## beta.defense[2] -0.2931 -0.148 -0.0741 -0.00196 0.138
## beta.home[1]   -6.3655 -4.514 -3.5414 -2.58559 -0.756
## beta.home[2]   -3.8301 -1.887 -0.8491  0.16783 2.062
## beta.home[3]   -3.6558 -1.769 -0.8066  0.17242 2.007
## beta.home[4]   -3.1788 -1.211 -0.1571  0.88799 2.859
## beta.home[5]   -6.6877 -4.098 -2.7803 -1.43849 1.176
## beta.home[6]   -2.4894  0.204  1.5977  3.00227 5.706
## beta.home[7]   -0.5994  2.064  3.4377  4.83181 7.517

```

```

## beta.home[8]      -0.6143    2.315   3.7747   5.25484   8.181
## delta[1]        -1978.4088 -688.037  -4.1092 662.68424 1965.061
## delta[2]        -1940.6629 -675.923  -2.0463 671.92029 1962.165
## eta[1]          -1413.8970 -482.575  -5.4483 474.92512 1370.112
## eta[2]          -1394.8328 -476.146   1.7122 481.47216 1388.076
## eta[3]          -1396.0830 -483.186  -9.9967 470.21005 1376.652
## eta[4]          -1382.7089 -479.828  -1.6793 479.87624 1378.586
## rho[1]          -1387.9001 -483.225  -2.6988 464.93505 1388.009
## rho[2]          -1382.8577 -480.359  -6.9693 470.32306 1373.230
## rho[3]          -1376.5466 -471.454  -6.5531 468.53942 1366.468
## rho[4]          -1383.8694 -474.965  -1.9381 476.58238 1379.023
## sigmasq         28.7819   30.779   31.8832   33.05067   35.406

```

Effective Sample Size

```
(eff.size = effectiveSize(result$coda.sam[, ]))
```

```

##   beta.away[1]   beta.away[2]   beta.away[3]   beta.away[4]   beta.away[5]
##      3711       3557       4146       3985       1718
##   beta.away[6]   beta.away[7]   beta.away[8]   beta.defense[1] beta.defense[2]
##      1736       1787       1849       2988       1497
##   beta.home[1]   beta.home[2]   beta.home[3]   beta.home[4]   beta.home[5]
##      3792       4202       3943       3788       1767
##   beta.home[6]   beta.home[7]   beta.home[8]   delta[1]     delta[2]
##      1773       1805       1817       29040      30000
##   eta[1]         eta[2]       eta[3]       eta[4]     rho[1]
##      30088      29956      30305      30853      29666
##   rho[2]         rho[3]       rho[4]       sigmasq
##      29489      30000      30441      29940

```

The effective sample sizes of all parameters are greater than 400.

Probabilty of players perform better at home than away

```

post.samp = as.matrix(result$coda.sam)

beta.home = post.samp[, paste("beta.home[", 1:(Num.Rank*Num.Position), "]")]
beta.away = post.samp[, paste("beta.away[", 1:(Num.Rank*Num.Position), "]")]

prob.home.away = rep(0, Num.Rank * Num.Position)
for (r in 1:Num.Rank) {
  for (p in 1:Num.Position) {
    idx = (p-1) * Num.Rank + r
    prob.home.away[idx] = mean(beta.home[, idx] > beta.away[, idx])
  }
}
prob.home.away

```

```
## [1] 0.8093 0.9147 0.5072 0.7559 0.6288 0.9248 0.9508 0.7928
```

```

prob.home.away.df = data.frame(colnames(X.home))
prob.home.away.df$prob.home.bt.away = prob.home.away

```

```

colnames(prob.home.away.df) = c("Rank:Position", "Prob.home.bt.away")

kable(prob.home.away.df)

```

Rank:Position	Prob.home.bt.away
RankRank1:PositionPK	0.8093
RankRank2:PositionPK	0.9147
RankRank3:PositionPK	0.5072
RankRank4:PositionPK	0.7559
RankRank1:PositionQB	0.6288
RankRank2:PositionQB	0.9248
RankRank3:PositionQB	0.9508
RankRank4:PositionQB	0.7928

Beta defense

If a player is facing a team which gives up more points to players on average, we expect the player will score more points.

```

Num.fixed.size=Num.Position*Num.fixed.pred
beta.defense = post.samp[, paste("beta.defense[", 1:Num.fixed.size, "] ", sep="")]

beta.defense.int.df = data.frame(colnames(X.defense))
beta.defense.int = matrix(rep(0, Num.fixed.size * 4), nrow=Num.fixed.size, ncol = 4)

int.alpha=0.05
for (p in 1:Num.Position) {
  for (f in 1:Num.fixed.pred) {
    idx = (f-1) * Num.Position + p
    beta.defense.int[idx, 1:3] = quantile(beta.defense[, idx], c(int.alpha/2, 0.5, 1-int.alpha/2))
    beta.defense.int[idx, 4] = mean(beta.defense[, idx])
  }
}

beta.defense.int$`pct025` = beta.defense.int[, 1]
beta.defense.int$`pct975` = beta.defense.int[, 3]
beta.defense.int$`median` = beta.defense.int[, 2]
beta.defense.int$`mean` = beta.defense.int[, 4]
colnames(beta.defense.int.df) = c("beta.defense.position", "pct025", "pct975", "median", "mean")
kable(beta.defense.int.df)

```

beta.defense.position	pct025	pct975	median	mean
AvgOppPAP7Wks:PositionPK	-0.0663	0.5435	0.2350	0.2347
AvgOppPAP7Wks:PositionQB	-0.2931	0.1377	-0.0741	-0.0750

We observe that the median beta.defense for PK is positive as expected. But for QB, it is negative, that implies QB actually scores less against bad defensive team.

DIC

```
(dic.samp = dic.samples(m, NTotalSim))
```

```
## Mean deviance: 4700
## penalty 19
## Penalized deviance: 4719
```

The effective number of parameters (“penalty”) is 19, and the Plummer’s DIC (“Penalized deviance”) is 4719. Not that we have 29 parameters in our model, 10 of them were shrunk away.

Model Checking

A posterior predictive p-value based on the no-intercept model. Consider test quantity

$$T(y, X, \theta) = |\hat{c}or(\epsilon, \text{time})|$$

where $\hat{c}or(\epsilon, \text{time})$ is sample correlation between the error vector ϵ and the year week in the data. The larger this quantity is for the model, the less well it fits the data (since, if a regression model actually fits, the errors should ideally be uncorrelated with the predictor).

The simulated error vectors ϵ (as rows of a matrix):

```
error.sim <- matrix(NA, NTotalSim, nrow(fdp_train))
y_hat.sim <- matrix(NA, NTotalSim, nrow(fdp_train))
for(s in 1:NTotalSim) {
  y_hat.sim[s, ] = fdp_train$AvgPts5Wks + (X.defense %*% beta.defense[s, ]) + (X.home %*% beta.home[s,
    error.sim[s, ] <- (fdp_train$FanDuelPts - y_hat.sim[s, ])
}
```

The simulated replicate error vectors ϵ^{rep} (as rows of a matrix), which are the error vectors computed using replicate response vectors y^{rep} :

```
post.sigma.2.sim <- post.samp[, "sigmasq"]
post.sigma.sim <- sqrt(post.sigma.2.sim)

yreps <- matrix(NA, NTotalSim, nrow(fdp_train))
for(s in 1:NTotalSim) {
  yreps[s, ] <- rnorm(nrow(fdp_train), y_hat.sim[s, ], post.sigma.sim[s])
}

error.rep <- matrix(NA, NTotalSim, nrow(fdp_train))

for(s in 1:NTotalSim) {
  error.rep[s, ] <- (yreps[s, ] - y_hat.sim[s, ])
}
```

The simulated values of $T(y, X, \theta)$

```
T.sim = abs(cor(t(error.sim), fdp_train$FanDuelPts))
head(T.sim)
```

```

##      [,1]
## [1,] 0.7236
## [2,] 0.6846
## [3,] 0.7125
## [4,] 0.6739
## [5,] 0.7069
## [6,] 0.7022

```

The simulated values of $T(y^{rep}, X, \theta)$

```

T.rep.sim = abs(cor(t(error.rep), fdp_train$FanDuelPts))
head(T.rep.sim)

```

```

##      [,1]
## [1,] 0.06680
## [2,] 0.09912
## [3,] 0.01110
## [4,] 0.01664
## [5,] 0.01113
## [6,] 0.03517

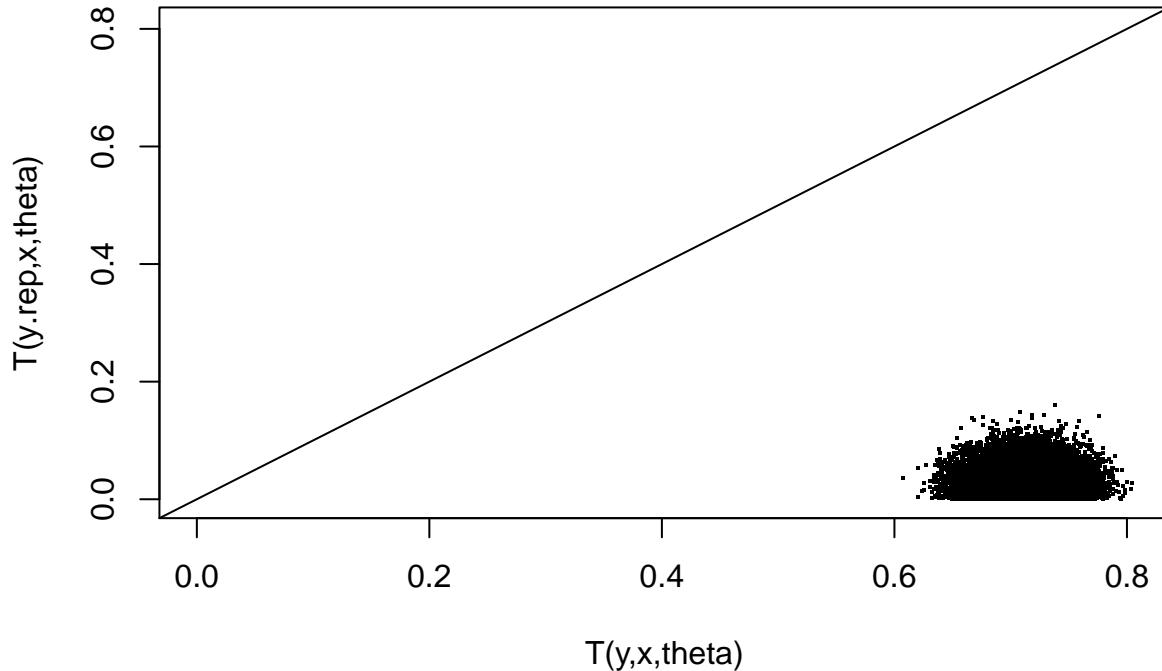
```

The simulated values of $T(y^{rep}, X, \theta)$ versus those of $T(y, X, \theta)$, with a reference line indicating where the two values would be equal.

```

plot(T.sim, T.rep.sim, pch=".",
      xlim=c(min(T.sim, T.rep.sim), max(T.sim, T.rep.sim)),
      ylim=c(min(T.sim, T.rep.sim), max(T.sim, T.rep.sim)),
      xlab="T(y,x,theta)", ylab="T(y.rep,x,theta)")
abline(a=0,b=1)

```



The posterior predictive p-value:

```
(p.value = mean(T.rep.sim >= T.sim))
```

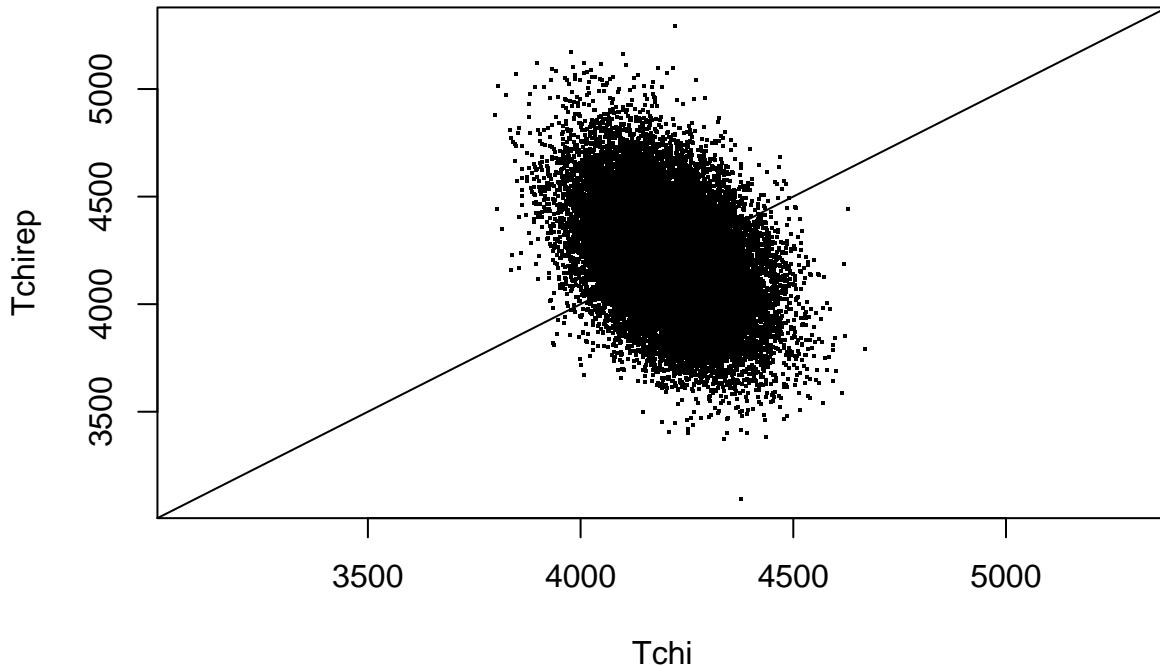
```
## [1] 0
```

The p.value is 0. This indicates the error actually has correlation with the response.

```
Tchi <- numeric(NTotalSim)
Tchirep <- numeric(NTotalSim)
for(s in 1:NTotalSim){
  Tchi[s] <- sum((fdp_train$FanDuelPts - y_hat.sim[s,])^2 / post.sigma.sim[s])
  Tchirep[s] <- sum((yreps[s,] - y_hat.sim[s,])^2 / post.sigma.sim[s])
}
(p.value.Tchi = mean(Tchirep >= Tchi))
```

```
## [1] 0.5057
```

```
plot(Tchi, Tchirep, pch=".",
  xlim=c(min(Tchi, Tchirep), max(Tchi, Tchirep)),
  ylim=c(min(Tchi, Tchirep), max(Tchi, Tchirep)),
  xlab="Tchi", ylab="Tchirep")
abline(a=0,b=1)
```



The posterior predictive p-value using the chi-square discrepancy is `p.value.Tchi=0.5057`. The p-value is > 0.05 . Hence, it does not indicate any evidence of problems.

Using $Pr(y^{rep} \geq y|y)$ as posterior predictive p-value

```

yreps.minus.y <- matrix(NA, NTotalSim, nrow(fdp_train))
for(s in 1:NTotalSim) {
  yreps.minus.y[s, ] <- yreps[s, ] - fdp_train$FanDuelPts
}

(p.value.y.rep.all = mean(yreps.minus.y > 0))

## [1] 0.5076

```

The posterior predictive p-value using individual data point is `p.value.y.rep.all = 0.5076`, which is > 0.05 . This shows no evidence of problem.

Prediction

```

if (Num.Position == 1) {
  #X.defense = model.matrix(~ 0 + AvgOppPAP7Wks + FanDuelSalary, data=fdp_train)
  X.defense.test = model.matrix(~ 0 + AvgOppPAP7Wks, data=fdp_test)
  if (Use.Rank) {
    X.home.test = model.matrix(~ 0 + Rank , data=fdp_test)
  }
}

```

```

    X.away.test = model.matrix(~ 0 + Rank , data=fdp_test)
} else {
  X.home.test = rep(1, nrow(fdp_test))
  X.away.test = rep(1, nrow(fdp_test))
}
} else {
#X.defense = model.matrix(~ 0 + AvgOppPAP7Wks:Position + FanDuelSalary:Position, data=fdp_train)
X.defense.test = model.matrix(~ 0 + AvgOppPAP7Wks:Position, data=fdp_test)
if (Use.Rank) {
  X.home.test = model.matrix(~ 0 + Rank:Position , data=fdp_test)
  X.away.test = model.matrix(~ 0 + Rank:Position , data=fdp_test)
} else {
  X.home.test = model.matrix(~ 0 + Position , data=fdp_test)
  X.away.test = model.matrix(~ 0 + Position , data=fdp_test)
}
}

X.home.test = X.home.test * fdp_test$HomeGame
X.away.test = X.away.test * (1- fdp_test$HomeGame)
X.test = cbind(X.defense.test, X.home.test, X.away.test)

y_hat.test <- matrix(NA, NTotalSim, nrow(fdp_test))
for(s in 1:NTotalSim) {
  y_hat.test[s, ] = fdp_test$AvgPts5Wks + (X.defense.test %*% beta.defense[s, ]) + (X.home.test %*% beta.home[s, ])
}

y.pred <- matrix(NA, NTotalSim, nrow(fdp_test))
for(s in 1:NTotalSim) {
  y.pred[s, ] <- rnorm(nrow(fdp_test), y_hat.test[s, ], post.sigma.sim[s])
}

```

Prediction of individual data point

Let's pick one example data point, `fdp_test[1,]`

```
(fdp_test[1,])
```

```

##      Position Year YearWeek Opponent Week PlayerId          Name      Team HomeGame
## 37        QB 2016     201615   Broncos  15     1131 Brady, Tom Patriots       0
##      FanDuelPts FanDuelSalary AvgOppPAP7Wks SdOppPAP7Wks OallAvgPAP OallStdevPAP
## 37        7.42           8100      12.24      6.266     17.44      2.909
##      AvgPts5Wks StdevPts5Wks OffRnk5Wks DefRnk7Wks   Rank
## 37        20.36          7.192      Rank1      Rank4 Rank1

```

The real FanDuelPts is 7.42

The prediction value has the following 95% interval

```
quantile(y.pred[, 1], c(0.025, 0.975))
```

```

##      2.5% 97.5%
##  4.944 27.266

```

which does contain the actual data value of 7.42

Let's look at $Pr(y_{pred} \geq y)$

```
y.pred.minus.y <- matrix(NA, NTotalSim, nrow(fdp_test))
for(s in 1:NTotalSim) {
  y.pred.minus.y[s, ] <- y.pred[s, ] - fdp_test$FanDuelPts
}

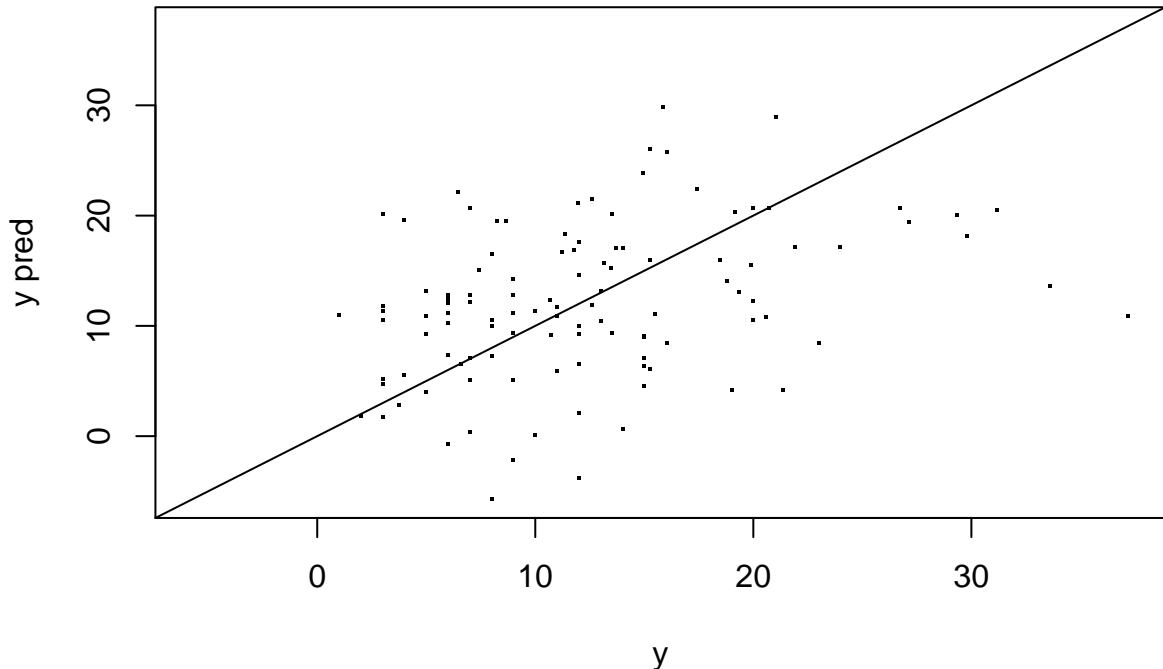
(p.value.y.pred.all = mean(y.pred.minus.y > 0))

## [1] 0.4683
```

The probability of $y_{pred} \geq y$ is 0.4683, close to 0.5, which implies a relatively good predictive value.

A look at a cross section of how one simulation of a prediction of the whole test set

```
for (s in 1:1) {
  plot(fdp_test$FanDuelPts, y.pred[s, ], pch=".",
    xlim=c(min(y.pred[s, ], fdp_test$FanDuelPts), max(y.pred[s, ], fdp_test$FanDuelPts)),
    ylim=c(min(y.pred[s, ], fdp_test$FanDuelPts), max(y.pred[s, ], fdp_test$FanDuelPts)),
    xlab="y", ylab="y pred")
  abline(a=0,b=1)
}
```



Alternative Model - no rank

```
#sink("fdp.norank.bug")
#cat("
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(alpha[i] + inprod(X.defense[i, ], beta.defense)
                  + inprod(X.home[i, ], beta.home)
                  + inprod(X.away[i, ], beta.away), sigmasqinv)
  }

  # The entry of the beta.defense corresponds to Opponent:Position
  # In our model, we pool the beta.defense based on position.
  # i.e. All defense effects of the same position are drawn from the same distribution
  for (p in 1:Num.Position) {
    beta.defense[p] ~ dnorm(delta[p], 1/1000^2)
    delta[p] ~ dnorm(0, 1/100000^2)
  }

  # The entry of the beta.home and beta.away corresponds to Position
  # In our model, we pool the beta.home/away based on Position
  # NO RANK
  for (t in 1:Num.Position) {
    beta.home[t] ~ dnorm(eta, 1/1000^2)
    beta.away[t] ~ dnorm(rho, 1/1000^2)
  }
  eta ~ dnorm(0, 1/100000^2)
  rho ~ dnorm(0, 1/100000^2)

  sigmasqinv ~ dgamma(0.0001, 0.0001)
  sigmasq <- 1/sigmasqinv
}
#      ",fill = TRUE)
#sink()
```

```
Use.Rank = FALSE
Num.Opponent = length(unique(fdp_train[, "Opponent"]))
Num.Position = length(unique(fdp_train[, "Position"]))
#Num.fixed.pred=2 #AvgOppPAP7Wks + FanDuelSalary
Num.fixed.pred=1 #AvgOppPAP7Wks
if (Use.Rank) {
  Num.Rank = length(unique(fdp_train[, "Rank"]))
  Num.HomeAwayInit = Num.Rank
  Model.File.Ext = ""
} else {
  Num.HomeAwayInit = 1
  Model.File.Ext = ".norank"
}

if (Num.Position == 1) {
  #X.defense = model.matrix(~ 0 + AvgOppPAP7Wks + FanDuelSalary, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks, data=fdp_train)
```

```

if (Use.Rank) {
  X.home = model.matrix(~ 0 + Rank , data=fdp_train)
  X.away = model.matrix(~ 0 + Rank , data=fdp_train)
} else {
  X.home = rep(1, nrow(fdp_train))
  X.away = rep(1, nrow(fdp_train))
}
} else {
  #X.defense = model.matrix(~ 0 + AvgOppPAP7Wks:Position + FanDuelSalary:Position, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks:Position, data=fdp_train)
  if (Use.Rank) {
    X.home = model.matrix(~ 0 + Rank:Position , data=fdp_train)
    X.away = model.matrix(~ 0 + Rank:Position , data=fdp_train)
  } else {
    X.home = model.matrix(~ 0 + Position , data=fdp_train)
    X.away = model.matrix(~ 0 + Position , data=fdp_train)
  }
}

X.home = X.home * fdp_train$HomeGame
X.away = X.away * (1- fdp_train$HomeGame)
X = cbind(X.defense, X.home, X.away)

```

```

# Initialization List for the 4 chains
jags.inits=list(
  list( sigmasqinv= 0.01, delta = rep(-100000, Num.Position * Num.fixed.pred),
        eta = c(100000, -100000, 100000, -100000)[1:Num.HomeAwayInit],
        rho = c(-100000, 100000, -100000, 100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 ),
  list( sigmasqinv= 0.01, delta = rep(100000, Num.Position * Num.fixed.pred),
        eta = c(100000, -100000, -100000, 100000)[1:Num.HomeAwayInit],
        rho = c(-100000, 100000, 100000, -100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 1 ),
  list( sigmasqinv=0.000001, delta = rep(-100000, Num.Position * Num.fixed.pred),
        eta = c(-100000, 100000, -100000, 100000)[1:Num.HomeAwayInit],
        rho = c(100000, -100000, 100000, -100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 2 ),
  list( sigmasqinv=0.000001, delta = rep(100000, Num.Position * Num.fixed.pred),
        eta = c(-100000, 100000, 100000, -100000)[1:Num.HomeAwayInit],
        rho = c(100000, -100000, -100000, 100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 3 )
)

data.jags <- list(
  y= fdp_train$FanDuelPts,
  alpha = fdp_train$AvgPts5Wks,
  X.defense = X.defense,
  X.home = X.home,
  X.away = X.away,
  Num.fixed.pred=Num.fixed.pred,
  Num.Position=Num.Position
  #Num.Opponent=Num.Opponent,

```

```

#Num.Rank=Num.Rank
}

runModel=TRUE
runSample=TRUE

mon.col <- c("delta", "eta", "rho", "beta.defense", "beta.home", "beta.away", "sigmasq")

NSim = 30000
NChain = 4
NThin = 5
NTotalSim = NSim * NChain / 5
if (runModel) {
  if(Use.Rank) {
    bug.file = "fdp.bug"
  } else {
    bug.file = "fdp.norank.bug"
  }
  m <- jags.model(bug.file, data.jags, inits = jags.inits, n.chains=NChain, n.adapt = 1000)
  save(file=paste("fdp.jags.model.init", Model.File.Ext, ".Rdata", sep=""), list="m")
} else {
  load(paste("fdp.jags.model.init", Model.File.Ext, ".Rdata", sep=""))
  m$recompile()
}

## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 746
##   Unobserved stochastic nodes: 11
##   Total graph size: 9356
##
## Initializing model

load.module("dic")

N.Retry.Loop = 1
if (runSample) {
  N.burnin=2500/2
  for (loopIdx in 1:N.Retry.Loop) {
    (start_time <- Sys.time())
    (N.burnin = N.burnin * 2)
    result = burnAndSample(m, N.burnin, NSim, show.plot=FALSE, mon.col = mon.col, n.thin=NChain)
    (end_time <- Sys.time())
    (result$gelman.R.max)
  }
  run.params = paste(".", N.burnin, ".", NChain, ".", NSim, ".", NThin, sep="")
  save(file=paste("fdp.jags.samples", run.params, Model.File.Ext, ".Rdata", sep=""), list="result")
  save(file=paste("fdp.jags.model", run.params, Model.File.Ext, ".Rdata", sep=""), list="m")
} else {
  N.burnin=2500/2 * (2**N.Retry.Loop)
  run.params = paste(".", N.burnin, ".", NChain, ".", NSim, ".", NThin, sep="")
}

```

```

load(paste("fdp.jags.samples", run.params, Model.File.Ext, ".Rdata", sep=""))
load(paste("fdp.jags.model", run.params, Model.File.Ext, ".Rdata", sep=""))

gelman.diag(result$coda.sam, autoburnin=FALSE, multivariate = FALSE)
}

```

```

## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## beta.away[1]      1     1.00
## beta.away[2]      1     1.01
## beta.defense[1]   1     1.00
## beta.defense[2]   1     1.00
## beta.home[1]      1     1.00
## beta.home[2]      1     1.00
## delta[1]          1     1.00
## delta[2]          1     1.00
## eta              1     1.00
## rho              1     1.00
## sigmasq          1     1.00

```

Converged as gelman.R.max = 1.0017 < 1.1 and the plot also looks good.

```
(m.summary = summary(result$coda.sam))
```

```

##
## Iterations = 2504:32500
## Thinning interval = 4
## Number of chains = 4
## Sample size per chain = 7500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean        SD Naive SE Time-series SE
## beta.away[1] -2.4868  1.374 0.007933  0.02422
## beta.away[2] -1.8561  2.094 0.012089  0.05400
## beta.defense[1] 0.2522  0.162 0.000933  0.00293
## beta.defense[2] 0.0178  0.114 0.000659  0.00299
## beta.home[1] -1.5692  1.352 0.007806  0.02381
## beta.home[2] -0.3188  2.044 0.011800  0.05241
## delta[1]      5.7158 1003.464 5.793504  5.82433
## delta[2]      -9.5228 1004.742 5.800878  5.80110
## eta           -1.4774 709.925 4.098752  4.07614
## rho           -3.8073 703.095 4.059322  4.07909
## sigmasq       35.2576  1.836 0.010603  0.01070
##
## 2. Quantiles for each variable:
##
##          2.5%    25%    50%    75%   97.5%
## beta.away[1] -5.1688 -3.4072 -2.4840 -1.5705  0.226
## beta.away[2] -5.9390 -3.2664 -1.8725 -0.4445  2.312

```

```

## beta.defense[1] -0.0665 0.1436 0.2530 0.3604 0.568
## beta.defense[2] -0.2074 -0.0592 0.0183 0.0946 0.240
## beta.home[1] -4.2070 -2.4793 -1.5684 -0.6617 1.090
## beta.home[2] -4.2767 -1.7074 -0.3297 1.0538 3.712
## delta[1] -1963.6012 -667.1541 2.2090 687.1508 1975.654
## delta[2] -1994.6211 -688.6346 -4.0685 670.0538 1930.427
## eta -1393.1374 -478.2809 -2.4320 476.9654 1390.406
## rho -1390.2689 -479.6002 -3.7107 470.8765 1368.090
## sigmasq 31.8231 34.0001 35.1910 36.4582 39.015

```

Effective Sample Size

```
(eff.size = effectiveSize(result$coda.sam[, ]))
```

	beta.away[1]	beta.away[2]	beta.defense[1]	beta.defense[2]	beta.home[1]
##	3228	1505	3051	1453	3238
##	beta.home[2]	delta[1]	delta[2]	eta	rho
##	1521	29696	30000	30402	29716
##	sigmasq				
##	29489				

The effective sample sizes of all parameters are greater than 400.

DIC

```
(dic.samp = dic.samples(m, NTotalSim))
```

```

## Mean deviance: 4774
## penalty 7.02
## Penalized deviance: 4781

```

The effective number of parameters (“penalty”) is 7.02, and the Plummer’s DIC (“Penalized deviance”) is 4781. This model has a higher DIC compared with the original one with rank(4719). Hence, we conclude that the original model (with rank) is better for prediction.

Conclusion

While the model has some issue with error correlation, it has decent prediction ability. It should be noted that we only picked two positions to predict due to computation resource constraint. In the pursue of creating this model, we have multiple route:

- 1) We have tried to include all positions. However, the model took 5 hours to finish the MCMC simulation. Therefore, we elected to include a smaller subset.
- 2) We have tried to include FanDuelSalary as a predictor. However, that does not improve the model, and added sufficient time to compute - About 30 minutes to compute.